Amendments to the Claims:

This listing of claims replaces all prior versions and listings of claims in the application:

Listing of Claims:

1. (Cancelled)

2. (Previously Presented) A computer-implemented vault for archiving software components, the vault comprising:

one or more software components stored in the vault;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault; and

a post controller for performing a direct, random access insertion of a software component to the vault wherein the post controller generates a unique key from the new component and optimizes storage if the unique key exists.

3. (Previously Presented) A computer-implemented vault for archiving software components, the vault comprising:

one or more software components stored in the vault;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault; and

a client coupled to the vault, the client having a physical software component residing on the client, the client generating a key from the physical software component.

4. (Currently Amended) A computer-implemented vault for archiving software components, the vault comprising:

one or more software components stored in the vault;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault;

one or more secondary vaults coupled to the vault; ~~and~~

a fault-tolerant rollover system for sequentially searching each vault for the presence of a target software component; and

a client for generating a key, the client applying the key to recover the target software component from the most accessible of the vaults.

5. (Previously Presented) The computer-implemented vault of claim 4, wherein the secondary vaults are ordered based on accessibility of the vaults.

6. (Canceled)

7. (Currently Amended) The computer-implemented vault of claim ~~6~~ 4, wherein the client uses a metadata description to generate the key.

8. (Currently Amended) The computer-implemented vault of claim ~~6~~ 4, wherein the search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

9. (Previously Presented) A computer-implemented vault for archiving software components, the vault comprising:

means for storing one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault; and

a post means for performing a direct, random access insertion of a software component to the vault wherein the post means generates a unique key from the new component and optimizes storage if the unique key exists.

10. (Previously Presented) A computer-implemented vault for archiving software components, the vault comprising:

means for storing one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault; and

a client coupled to the vault, the client having a physical software component residing on the client, the client generating a key from the physical software component.

11. (Currently Amended) A computer-implemented vault for archiving software components, the vault comprising:

means for storing one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault;

one or more secondary vaults coupled to the vault; ~~and~~

means for sequentially searching each vault for the presence of a target software component; and

a client for generating a key, the client having a means for applying the key to recover the target software component from the most accessible of the vaults.

12. (Previously Presented) The computer-implemented vault of claim 10, wherein the secondary vaults are ordered based on accessibility of the vaults.

13. (Canceled)

14. (Currently Amended) The computer-implemented vault of claim 11~~-13~~, wherein the client uses a metadata description to generate the key.

15. (Currently Amended) The computer-implemented vault of claim 11~~-13~~, wherein the search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

16. (Previously Presented) A computer-implemented method for archiving software components, the method comprising:

storing one or more software components in a vault; and

performing a direct, random access retrieval of the one or more software components from the vault; and

performing a direct, random access insertion of a software component to the vault wherein said step of performing an insertion generates a unique key from the new component and optimizes storage if the key exists.

17. (Previously Presented) A method for archiving software components, the method comprising:

storing unique instances of the one or more software components in a vault;

performing a direct, random access retrieval of the one or more software components from the vault; and

generating a key from a physical software component residing on a client coupled to the vault.

18. (Currently Amended) A method for archiving software components, the method comprising:

storing unique instances of the one or more software components in a vault that is coupled to one or more secondary vaults;

performing a direct, random access retrieval of the one or more software components from the vault; ~~and~~

sequentially searching each vault for the presence of a target software component; and

generating a key that can be applied to recover the target software from the most accessible of the vaults.

19. (Previously Presented) The method of claim 18, wherein the secondary vaults are ordered based on accessibility of the vaults.

20. (Canceled)

21. (Currently Amended) The method of claim ~~20~~ 18, wherein ~~the client uses~~ a metadata description is used to generate the key.

22. (Currently Amended) The method of claim ~~20~~ 18, wherein ~~the~~ a search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

23. (Previously Presented) A computer-implemented method for managing one or more software components of a software application, the method comprising:

analyzing run-time states of an application that includes one or more components and generating, for each component of the application and based on a result of analyzing run-time states, current metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

creating a first key for each of the one or more components of the application, each first key of a component being created from and unique to the current metadata that describes the component;

creating a second key for each of the one or more components of the application, each second key being created from and unique to metadata that was previously generated in conjunction with a previous determination of run-time states of the application; and

for each first key, comparing the first key to the second keys and, if the first key does not match any of the second keys, storing the first key's corresponding component on one of the software vaults by performing a direct random-access storage operation.

24. (Previously Presented) The method of claim 23, wherein performing a storage operation comprises:

storing the first key along with the first key's corresponding component in the first-accessed software vault.

25. (Previously Presented) The method of claim 23, wherein:

generating metadata for a component includes generating information about the size, name, and attributes of the component, and

creating a key includes verifying the integrity of the respective software component and generating an integrity checksum, and incorporating into the unique key the integrity checksum as well as information about the size, name and attributes of the respective software component.

26. (Previously Presented) A computer-implemented method for retrieving one or more software components of a software application, the method comprising:

analyzing run-time states of an application that includes one or more components and generating, for each component of the application and based on a result of analyzing run-time states, metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

creating a key for one of the one or more components of the application, the component being one that is to be accessed from the one or more software vaults, the key being created from and unique to the metadata that describes the component and including location attributes;

using the key to look up the software component sequentially on the one or more software vaults; and

accessing and retrieving the software component from a first software vault on which ~~it~~ the component is found.

27. (Previously Presented) A computer-implemented method for locating one or more software components of a software application, the method comprising:

analyzing run-time states of an application that includes one or more components and generating, for each component of the application and based on a result of analyzing run-time states, metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

creating a key for one of the one or more components of the application, the component being one that is to be located in the one or more software vaults, the key being created from and unique to the metadata that describes the component and including location attributes;

determining an order of accessibility for the software vaults;

for each software vault, using the location of the software vault and the key, forming a uniform resource locator (URL); and

looking-up the URL in the software vaults, based on the order of accessibility, until the is located.

28. (Previously Presented) A computer-implemented method for retrieving software components of a software application, the method including comprising:

analyzing run-time states of an application that includes one or more components and generating, for each component of the application and based on a result of analyzing run-time states, metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

transforming, for each of the one or more components, the metadata that describes the component into a key that that includes location attributes of the component; and

using the location attributes of each key, and retrieving the software components from one or more software vaults accessible to the client computer through the communications network.

29. (Previously Presented) The method of claim 28, further comprising:

determining an order of accessibility of the software vaults and retrieving the software components from the most accessible software vaults.

30. (Previously Presented) A computer-implemented method for recreating a software application, the method comprising:

analyzing run-time states of an application that includes one or more components and generating, for each component of the application and based on a result of analyzing run-time states, metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

transforming, for each of the one or more components, the metadata that describes the component into a first key that that includes location attributes of the component;

determining if the software components consume a large amount of file space and, upon a determination that the software components do consume a large amount of file space:

looking up, on the client computer, a second key for each of the one or more components;

for each of the one or more components, comparing the first key to the second key and, if the first and second keys do not match, retrieving the components from one or more remotely accessible software vaults, and, if the first and second keys do match, retrieving the component from the client computer; and

using the retrieved software components to recreate the software application.

31. (Previously Presented) The method of claim 30, comprising:

initializing a difference flag;

determining whether first binary data associated with the software components stored on the client computer and second binary data associated with the software components stored on the remotely accessible software vaults differ;

if the first and second binary data differ, comparing the first and second keys based on sequence attributes including date created, date modified, date last accessed or version number;

if the sequence attributes are equal, if one of the sequence attributes is newer than another, or if one of the sequence attributes is older than the other and the software components with the older attribute may be overwritten, setting the difference flag; and

determining that there is not a match between the first key and the second key if the difference flag is set.

32. (Previously Presented) A computer program product tangibly stored on machine-readable medium, the product comprising instructions to:

determine current run-times state of an application and generating, for each component of the application and based on a result of analyzing run-time states, metadata that describes the component; and

generate a key for each component of the application, the key including a hash of the metadata, wherein a same key is generated for the same metadata; and

compare a first key for a first of the application's components that is stored on a client computer with a second key for a second of the application's components that is stored on a storage device that is remote from the client computer, the keys being generated from metadata generated by determining run-time states of the application, and determine that the first and second of the application's components are the same if the keys match.

33. (Previously Presented) A computer program product tangibly stored on machine-readable medium, the product comprising instructions to:

analyze run-time states of an application that includes one or more components and generate, for each component of the application and based on a result of analyzing run-time states, current metadata that describes the component, the analyzing and generating being performed by a client computer, and the components being stored on one or more software vaults associated with and remote from the client computer;

create a first key for each of the one or more components of the application, each first key of a component being created from and unique to the current metadata that describes the component; and

for each first key, compare the first key to the second keys and, if the first key does not match any of the second keys, store the first key's corresponding component on one of the software vaults by performing a direct random-access storage operation.